Project 2: SARSA(λ) on the mountain-car problem

Kristian Aurlien and Mateusz Paluchowski
EPFL, Switzerland

***Abstract*—This paper is part of the mini project on unsupervised and reinforcement learning in neural networks conducted at EPFL in fall of 2016. It's purpose is to present process of analyzing and implementing SARSA(λ) algorithm for solving the mountain car problem.**

I. Introduction

State action rewards state action (SARSA) is an on-policy reinforcement learning algorithm used for learning a Markov decision process policy. SARSA(λ) is a modified version of the original algorithm where eligibility traces are introduced in order to reach the algorithm's convergence in a faster manner. In this mini project, an under-powered car has to find its way up a steep hill. Since the car does not have the sufficient power to directly climb the hill, the solution of the problem is to swing back and forth, applying the modest force provided by the cars engine in the right direction. The point of this project is to train neural network implementing SARSA(λ) such that the mountain car problem is solved. Influence of various SARSA(λ) parameters is also analyzed.

II. Main Part

A. Baseline

We start with a baseline agent which simply chooses the car's action completely randomly at each time step. Surprisingly this yields positive result more frequently than expected. For a 10 separate trials of 10 independent agents, thus 100 independent trials, the goal is reached 60 times which is 60% of all trials. However, it is worth noting that each trial was limited at quite significant number of time steps available - 2000 to be exact and each successful trial took 1409.14 time steps on average. Moreover, at each trial mountain car is reset at random location within the landscape and thus can start with non-zero potential energy, which can significantly contribute to achieving the goal. This and large number of time steps result in non-zero probability of reaching the final goal even for such trivial agent as random action selector.

B. SARSA(0)

Our second learning agent is neural network with a regular implementation of SARSA algorithm without eligibility traces. Just as suggested in the project description, we arrange input neurons in 20 by 20 grid, where the centers are distributed in the input space (-150,30) for the x-axis (position), and (-15,15) for the dx-axis (velocity). Each input neuron has a Gaussian activation function. Adjusting the variances so as to cover the whole input space, we get the following variances σ_x^2 and σ_{dx}^2:

$$\sigma_x^2 = (\frac{(150+30)}{20})^2 = 81$$

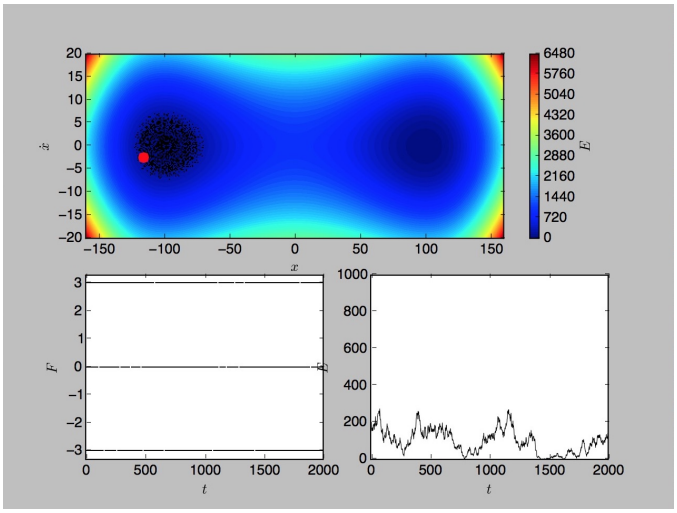

Fig. 1. Visualization of agent taking random actions and failing.

$$\sigma_{dx}^2 = (\frac{(15+15)}{20})^2 = 2.25$$

Just as expected, our neural network with simple SARSA() algorithm achieves better results than baseline random agent, however it still requires both quite substantial amount of time steps and number of trials in order to learn how to achieve the end goal. We decided to skip thorough analysis of this algorithm, simply due to the fact it takes significant amount of time to train to convergence.

C. SARSA(λ)

The eligibility trace enhanced version called SARSA(λ) provides best results in significantly shorter learning period. As we can see in the Figure 2, if the algorithm's various parameters are set just right, in average it converges to to a steady state in as low as 40 episodes and achieves its goal in less than 100 time steps. The quick learning is an effect of the eligibility traces and the exponentially decreasing τ. The eligibility trace helps pushing the reward received in the end back to the preceding actions, while an exploration-temperature τ that is initially high and decreasing with time, which helps the model with exploring in the beginning, and then gradually move towards always choosing the best expected action. In the next section we will see this more clearly, when we explore the effect of each of these parameters.

D. Model analysis

1) Learning curve: Figure 2 shows the learning curve of our final SARSA(λ) model, where we have averaged over 10

independent agents to obtain a smoother and more realistic model. For one agent the curve has a similar form, but the spikes are higher. We see that the model converges to a steady state after about 40 epochs, after which each epoch is completed in about 100 steps. Optimal parameters are as follows:

$$\tau = 1$$

$$\eta = 0.05$$

$$\gamma = 0.99$$

$$\lambda = 0.95$$

with exponential decay of τ towards 0.0005. Our optimal SARSA(λ) agent can reach the goal even below 50 time steps, which is presented in the Figure 3.

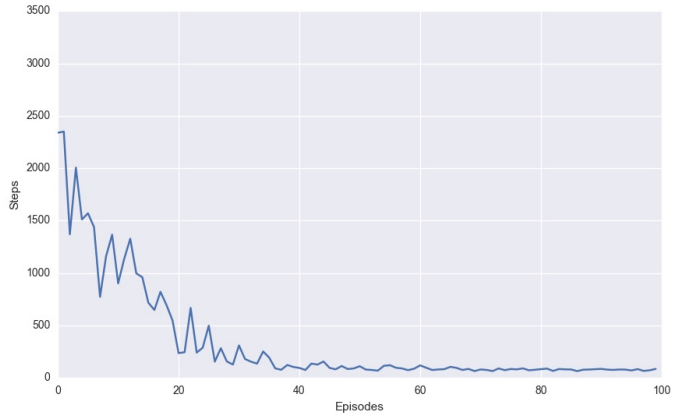

Fig. 2. Learning curve of SARSA(λ) with optimal parameters.

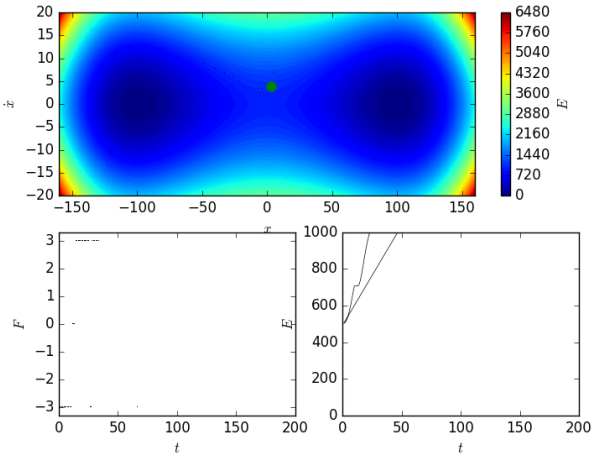

Fig. 3. Visualization of our optimal SARSA(λ) agent reaching its goal.

2) Temperature (τ): The ratio between exploration and exploitation of the model is controlled by the τ parameter. A big τ gives a model that will more often explore unseen actions, while a low τ will more often choose the action with the best expected reward. In the Figure 4, we have plotted the learning rate for four agents with a static τ of values $[0, \infty, 1]$, and one agent with exponential decay in the range $[1, 0.005)$.

With $\tau = 0$ there is no exploration, the agent always chooses the value with the highest expected value. As seen in the figure, this agent does not learn a good solution. $\tau = \infty$ on the other hand, will always explore and thus almost never takes the action that it expects to be good. Also this model fails to learn in reasonable time. Setting τ to 1 is a better choice, the learning curve now shows that the average completion time goes down. However, the fact that the car will still do quite a bit of exploration even after learning, leads to many spikes even after a long time. The last observation motivates the use of a decaying τ. This way we can start with a model that explores the action space in the beginning, but after learning what is best, it almost always chooses the action it has learned to be the best.

The exponential decay function we used for τ is described by $\tau_{final} + \tau_0 * e^{(-t/1000)}$, where τ_0 is the start value, and τ_{final} is the lower bound. For our model we found $\tau_0 = 1$ is the start value, and $\tau_{final} = 0.0005$ to give good results.

3) Eligibility trace decay rate (λ): The eligibility trace decay rate decides how quickly the value of a reward decreases with time. Setting λ to zero, the model only takes into account the current reward, while setting $\lambda = 1$ makes the model take into account all feature rewards. A good value for λ will speed up the learning, because also the previous actions leading the model to the reward state gets some reward. The learning curve plotted in the Figure 5 shows this in action. We see here that $\lambda = 0.95$ yields significantly better results than the other values.

4) Initial weights: Setting the initial weights to 1 and setting τ to zero corresponds to the "optimistic q-values" and "greedy policy" approach, which will lead to more exploration in the beginning. Using the exponential decaying τ we see that the models converge at about the same time with both weight initial values, but that by having the weights being set to zero in the beginning leads to more variance early on. One explanation to this might be that the optimal weights are closer to 1 than zero, so that the errors (and gradients) will be smaller with 1 as initial value.

5) Vector Fields: In order to visualize the behavior of the agent and thus the policy it learned we present the vector fields for three different trials in the Figure 7. As we can see the 'entropy' of vector fields decreases as the SARSA(λ) algorithm converges to stable solution. It is worth noting that policy seems to be energy preserving since for the hundredth trial we can observe multiple states for which optimal action is no action whatsoever. Moreover, first and twentieth trial seem to have much more contradictory actions where neighborhood states have opposite actions as their optimal action.

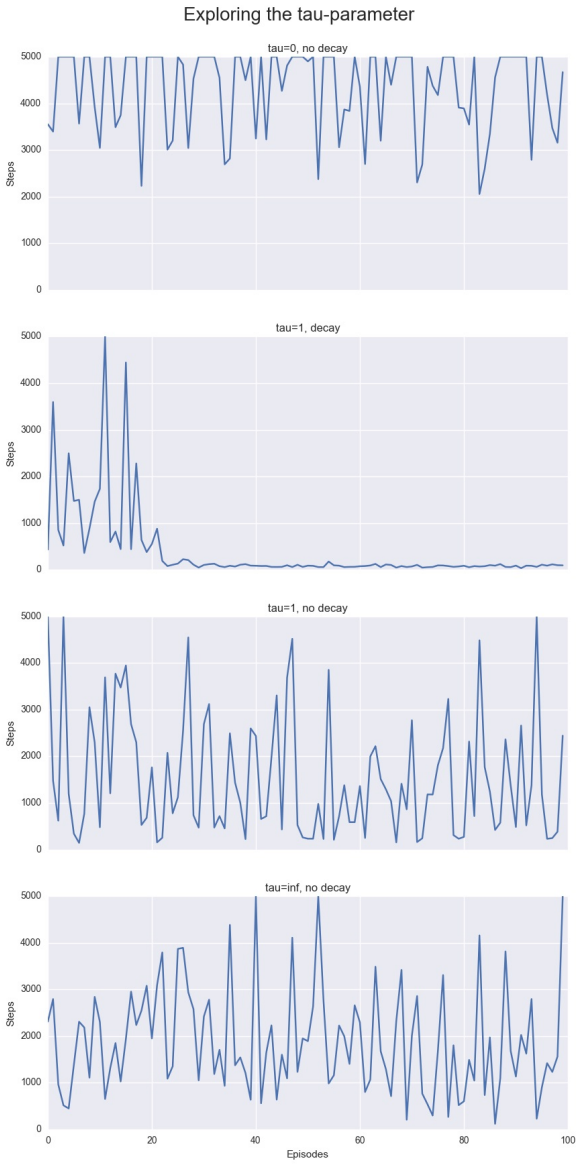

Fig. 4. Learning curves for different τ values

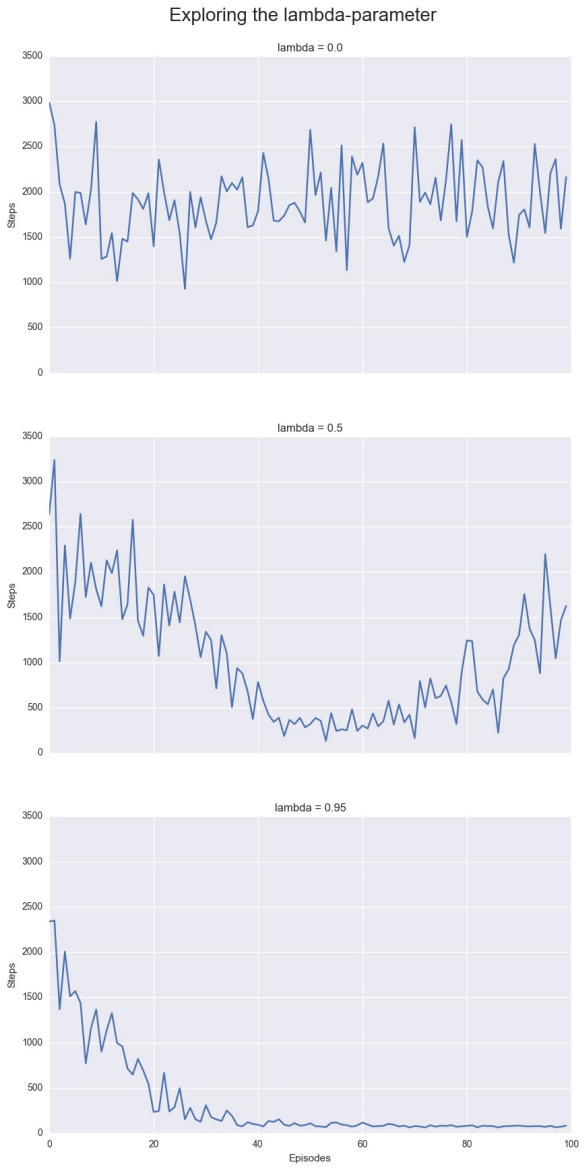

Fig. 5. Learning curves for different λ values

III. Conclusion

SARSA(λ) algorithm implemented as neural network is a really effective way of solving this problem, which is defined as Markov process. Nevertheless, we show that finding optimal parameters of the algorithm, such that it converges to stable solution in fastest manner, is not a trivial task. Eligibility trace version of SARSA clearly outperforms standard SARSA algorithm in terms of quick convergence towards acceptable solution. Last but not least, exponential decay of τ parameter seems to give a reasonable trade-off between initial exploration and final exploitation of available actions.

References

[1] Sutton, R. S., Barto, A. G., Reinforcement Learning: An Introduction. MIT Press, 1998.

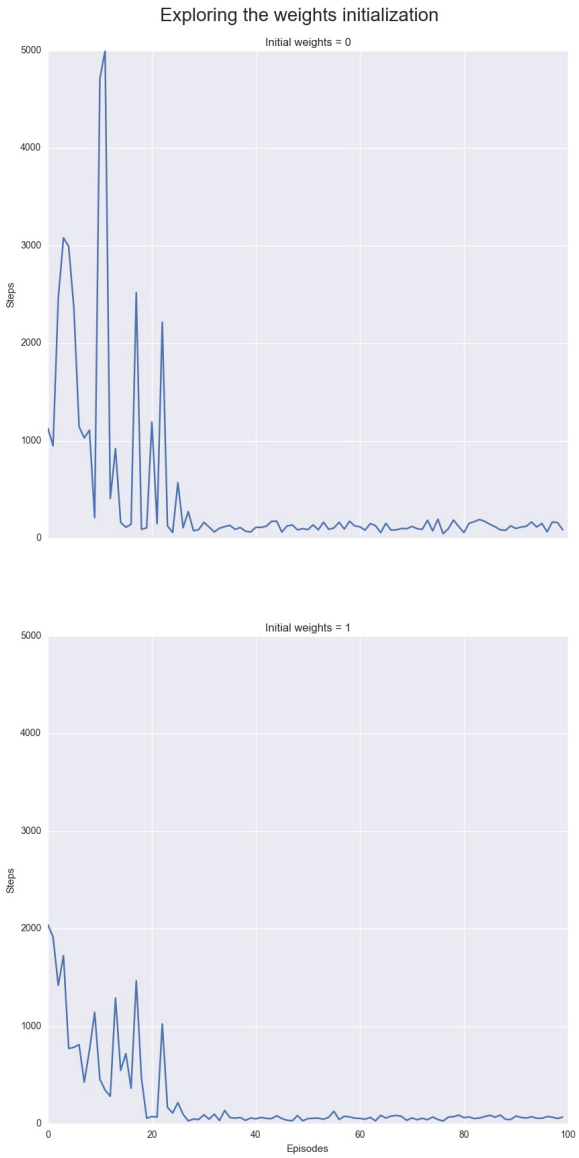

Fig. 6. Learning curves for different initial weights

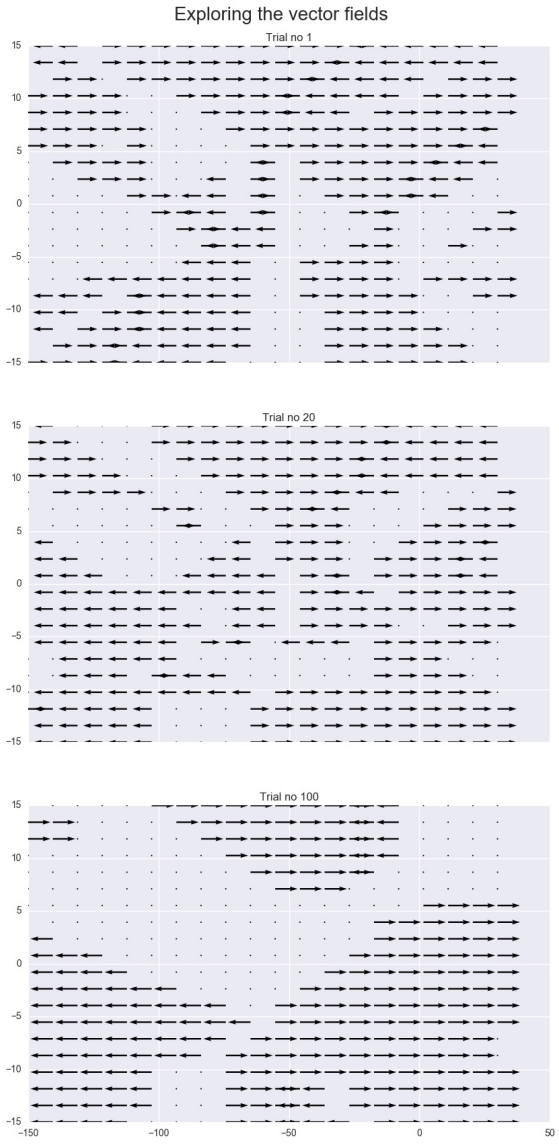

Fig. 7. Vector fields visualization for different trials.